BURN AFTER READING

A short guide to One-Time Pad Encryption Using Dice

LIKE THIS! --->

Vernam Cipher / One-time-pad

The Vernam Cipher or One-time-pad (OTP) is an encryption technique which does not always rely on computers and, if performed properly, is **unbreakable**.

1. Generating the key: generating the key (the one-time-pad) can be done in many ways, but one of the easiest ways which ensures (close to) true randomness is by rolling some **10-sided dice** (d10).

You can buy polyhedral dice at your local games/comic shop, or buy a bag of 50 d10s online.

Roll the dice as many times as it takes to obtain a key of the desired length. Write the key on a piece of graph paper. **Make only as many copies of the key as is needed to communicate.** Multiple keys can be generated in one session and put into a notepad.

NOTE on random number generators: There are many kinds of random number generators (**RNGs**). Generally, there are 2 categories. There are **True** random number generators (**TRNGs**) and there are **Pseudorandom** number generators (**PRNGs**).

TRNGs are **hardware** devices that rely on a **natural** source of randomness such as dice, a complete deck of properly shuffled cards, ambient noise, the avalanche breakdown of a diode, etc. There are even TRNGs that rely on the known rate of radioactive decay of certain materials.

PRNGs are usually software based. Examples of PRNGs include Google's random number generator, dice roller programs, and the /dev/random file built into Linux systems.

Randomness is an interesting mathematical concept that's more complex than we can get into here. In all honesty, True randomness is actually very hard to obtain. In most cases the best you can hope for is "good enough". But while PRNGs may be "good enough" for most applications like playing tabletop RPGs with your friends, it's important to try to do Better than a program running on a computer or a network which may be compromised or whose code may be flawed.

2. Encrypt your message: Let's say you want to encrypt the message "hello world" and your key is as follows:

85228 73961 05012 26896 59691 11361 30552 93412 49142 62259

We assign two-digit numerical values to each letter of the alphabet, with 01-26 being A-Z, and 00 being space.

```
00 01 02 03 04 05 06 07 08
 _  A  B  C  D  E  F  G  H

09 10 11 12 13 14 15 16 17
 I  J  K  L  M  N  O  P  Q

18 19 20 21 22 23 24 25 26
 R  S  T  U  V  W  X  Y  Z
```

(numbers 27-99 can be assigned other values such as lower-case letters, punctuation marks or whole common words **if you like**, but let's keep it simple for this example.)

So the numerical value of the message "HELLO WORLD" becomes:

08 05 12 12 15 00 23 15 18 12 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

To encrypt the message, we add the digits of each letter to the corresponding digits in the key, modulo 10 (%10). All this means is that if the number we end up with is greater than 10, we drop the first digit and only write down the second digit.

H = 08
0 + 8 = 8
8 + 5 = 13
13 mod 10 = 3
encrypted "H" = 83

Repeat the process for the next letter in the message:

E = 05
0 + 2 = 2
5 + 2 = 7
encrypted "E" = 27

L = 12
1 + 8 = 9
2 + 7 = 9
encrypted "L" = 99

```
    L = 12
    1 + 3 = 4
    2 + 9 = 11
    11 mod 10 = 1
    encrypted "L" = 41
```

If we continue this process until the end of the message, we end up with

83 27 99 41 76 05 24 37 76 08 53 69 11 13 61 30 55 29 34 12 49 14 26 22 59

as our encrypted message.

Key = 85 22 87 39 61 05 01 22 68 96 59 69 11 13 61 30 55 29 34 12 49 14 26 22 59

text = 08 05 12 12 15 00 23 15 18 12 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00

ciphertext = 83 27 86 38 76 05 24 37 76 08 53 69 11 13 61 30 55 29 34 12 49 14 26 22 59

This is where you send the ciphertext to the recipient, and destroy the key so that it can't be used again.

3. Decrypt the message: In order to decrypt the message, we do the same process in reverse. So we subtract the key from the ciphertext digit by digit. If the difference is a negative number, we modulo 10 in the opposite direction and add 10 to get the correct number.

```
ciphertext = 83
       key = 85
8 - 8 = 0
3 - 5 = -2 (or if the number is smaller than what is subtracted from
it, add 10)
-2 +10 = 8
decrypted 08 = "H"

ciphertext = 27
       key = 22
2 - 2 = 0
7 - 2 = 5
decrypted 05 = "E"
```

```
ciphertext = 99
       key = 87
9 - 8 = 1
9 - 7 = 2
decrypted: 12 = "L"
```

And continue the process until the end of the ciphertext. The result should be:

```
decrypted = 08 05 12 12 15 00 23 15 18 12 04 00 00 00 00 00 00 00 00
00 00 00 00 00 00
     text = H  E  L  L  O     W  O  R  L  D
```

Tips:

1. **Keep messages** as **short** as possible. Since the messages will be encrypted and decrypted **by hand**, unnecessarily **long messages** will be very tedious to deal with.

2. If the entire message is shorter than the length of the key, it is best practice to send the rest of the key as well. They will return zeros when decrypted, but will look like random numbers to anyone trying to crack the cipher. This is fine since you'll never use the same key again, and no information can be gleaned just from the length of the message.

3. It's also a good idea to begin the message with an identifier such as "key n" or use the first 5 digits of the key itself (and start the message on the *next* set of digits) to let the recipient know which key from the one-time-pad they should be using.

Conclusion: Vernam Ciphers are a slow, tedious, low-bandwidth encryption tool that requires practice and discipline. But the trade-off is unbreakable encryption. When American and Soviet spies used this type of communication during the Cold War, the encryption was cracked because spies were either lazy or low on resources. They tried to save paper by reusing keys. This allowed opponents to use frequency analysis on the intercepted messages, and eventually extract the keys which would allow them to decipher future messages.

Do not neglect the **ONE-TIME** aspect of the One-time Pad. Hence the title of this document, “Burn After Reading”.

There is no app for this, not really. If this is the type of communication you find to be necessary, it should be a foregone conclusion that you can’t trust your computing devices with the information you’re sending. As the saying goes, we’re all we’ve got.

/eof